

Projective Splitting Algorithms for Integer Linear Programming

Part 1: Pure Integer Programs

Federico Rodes*

Isabel Mendez-Diaz[†]

Paula Zabala[‡]

Abstract

We propose a new exact approach for solving integer linear programming (ILP) problems which we will call projective splitting algorithms (PSAs). Unlike classical methods for solving ILP problems, PSAs conduct the search for the optimal solution by generating candidate solutions tailored to specific values of the objective function. As a consequence of this strategy, the number of variables in the original ILP problem is systematically reduced without adding any additional constraint to the initial formulation.

This is the first of a two-part series on PSAs. In this paper we focus on the resolution of pure integer linear programming (PILP) problems, leaving the treatment of mixed integer linear programming (MILP) formulations to the second part of this series. The proposed algorithm was tested against the IBM ILOG CPLEX [2] optimizer on instances of the 0-1 Multidimensional Knapsack Problem (0-1MKP), showing satisfactory results on instances with a large number of variables.

1 Introduction

Linear programming (LP) [3, 4] is a mathematical modelling technique designed to optimize a linear function (objective function) of non-negative continuous variables (decision variables), while satisfying a system of linear equations or inequalities (constraints). A LP model that restricts some of the variables so that these take only non-negative integer values is known as MILP. When all variables are integer-constrained, we have a PILP model. We will use the term ILP to refer to any of the two types of problems mentioned above.

Many practical situations can be modelled as LP problems where decision variables must take on integer values. Generating good timetables, determining optimal schedules for jobs which are to be processed in a production line, designing efficient communication networks, container loading, determining efficient vehicle routes, and various problems arising in computational biology are a few examples.

From a practical point of view, most of the examples mentioned above are extremely difficult to solve. In theoretical computer science, this is captured by the fact that many ILP problems are classified as NP-hard [6] problems. Thus, because of the inherent difficulty and the enormous practical importance of NP-hard ILP problems, a large number of techniques have been proposed to solve them. The available techniques can roughly be classified into two main categories: *exact* and *heuristic* algorithms. Exact algorithms are guaranteed to find an optimal solution and to prove its optimality for every instance of an ILP problem. The run-time, however, often increases dramatically with the problem instance's size, and often only small or moderately-sized instances can be practically solved to proven optimality. For larger instances, the only possibility is usually to turn to heuristic algorithms that trade optimality for run-time, i.e., they are designed to obtain good but not necessarily optimal solutions in a reasonable amount of time.

The aim of this paper is to propose—to the best of our knowledge—a new exact algorithm for solving PILP problems. The algorithm will be called *PSA-pilp*, and the idea behind it is to decompose the initial PILP problem into simpler one-dimensional subproblems, and then to use that information to generate a finite number of candidate solutions tailored to each of the possible optimal objective values of the problem. The optimal solution is then found by examining the set of candidate solutions arising from the previous analysis. The second part of this series is intended to extend this methodology to the class of MILP formulations.

*Licentiate in Applied Mathematics, FCEyN, Universidad de Buenos Aires, Argentina. E-mail: rodesf@gmail.com

[†]Departamento de Computacion, FCEyN, Universidad de Buenos Aires, Argentina. Tel.: +54-11-4576-3390, Fax: +54-11-45763359, E-mail: imendez@dc.uba.ar

[‡]Departamento de Computacion, FCEyN, Universidad de Buenos Aires/CONICET, Argentina. Tel.: +54-11-4576-3390, Fax: +54-11-45763359, E-mail: pzabala@dc.uba.ar

The remainder of this paper is organized as follows. In the next section, we give a short overview of ILP techniques and introduce some notation. In Section 3, we present the basic concepts involving the PSA-*pilp* algorithm and account for the main steps of the method through the solution of a simple example. Section 4 is devoted to present the scheme of the PSA-*pilp* algorithm and to prove its convergence. Computational results on instances of the 0-1MKP are reported in Section 5. Finally, Section 6 summarizes the main achievements of the proposed approach and outlines some interesting directions for future research.

2 Integer linear programming, an overview

This section gives a short overview of the main concepts in integer programming. For an in-depth coverage of the subject we refer to the books on linear optimization by Chvátal [1], and on combinatorial and integer optimization by Wolsey [13] and Nemhauser and Wolsey [12].

2.1 LP and ILP formulations

LP is a class of optimization problems that involves non-negative continuous variables, an objective function linearly depending on the variables, and a set of constraints expressed as linear inequalities. We consider the form

$$\begin{aligned} (\mathbf{LP}) \quad & \text{maximize } z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h \\ & \text{subject to } \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{R}_+^n \end{aligned} \tag{1}$$

where $\mathbf{c} \in \mathbb{Z}^n$, $h \in \mathbb{Z}$, $\mathbf{A} \in \mathbb{Q}^{n \times m}$ and $\mathbf{b} \in \mathbb{Q}^m$ are data. A *feasible solution* to (1) is a vector $\mathbf{x} \in \mathbb{R}_+^n$ satisfying the condition $\mathbf{Ax} \leq \mathbf{b}$. The aim of this problem is to find a feasible solution that maximizes the objective function $z(\mathbf{x})$.

As mentioned before, if we restrict some of the variables of a LP problem to take on integer values we obtain an ILP problem. We consider the form

$$\begin{aligned} (\mathbf{ILP}) \quad & \text{maximize } z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h \\ & \text{subject to } \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p} \end{aligned} \tag{2}$$

where \mathbf{c} , h , \mathbf{A} and \mathbf{b} are defined as in (1). Without loss of generality, we assume that the variables indexed 1 through p , $p \leq n$, are the integer-constrained variables (the integer variables), and the variables indexed $p+1$ through n are called the continuous variables.

Throughout this work it will be assumed for simplicity that the feasible regions of (1) and (2) are bounded. In addition, we will denote by $z_{\mathbf{LP}}$ (resp. $z_{\mathbf{ILP}}$) the optimal objective value of the problem, and by $\text{cod}(z(\mathbf{x}))$ the codomain of $z(\mathbf{x})$ for the problem under consideration. Finally, let us note that, in the context of PILP problems, the assumption made about the objective function (we do not loss generality) automatically implies that $\text{cod}(z(\mathbf{x})) \subseteq \mathbb{Z}$. The utility of this observation will become clear in Section 3.

2.2 LP-relaxation

One of the most important concepts in ILP are *relaxations*, where some or all constraints of a problem are loosened or omitted. Relaxations are mostly used to obtain related, simpler problems which can be solved efficiently yielding bounds for the original problem.

The *linear programming relaxation* of the ILP problem (2) is obtained by relaxing the integrality constraint, i.e., replacing $\mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$ with $\mathbf{x} \in \mathbb{R}_+^n$, yielding the LP problem (1). Large instances of such LP problems can be efficiently solved in practice by using simplex-based algorithms [3, 4], interior-point methods [9] or column generation approaches [1]. As the feasible points of an ILP problem form a subset of the feasible points of its LP-relaxation, the optimal value of the LP-relaxation provides an upper bound on the optimal value of the original ILP problem. Therefore, if an optimal solution to the LP-relaxation satisfies the integrality restrictions, then that solution is also optimal for the ILP problem.

2.3 Exact algorithms

When considering exact approaches, the following methods have had significant success. See e.g. [7, 10, 13] for a general introduction to these mathematical programming techniques.

Cutting plane approach

When modelling integer optimization problems as ILP problems, an important goal is to find a strong formulation, for which the LP-relaxation provides a solution which lies in general not too far away from the integer optimum. For many such problems it is possible to strengthen an existing ILP formulation significantly by including further inequalities, preferably, facets of the convex hull of feasible solutions.

The general cutting plane approach relaxes initially the integrality restrictions of the original ILP problem and solves the resulting linear program. In case the resulting LP solution satisfies the integer requirements, this is the solution to the integer program; otherwise, the LP-relaxation can be tightened by adding an extra constraint which is satisfied by all feasible integral solutions but is violated by the current LP optimal solution. Such a constraint is called a *cut* or *cutting plane*. The new LP-relaxation is then resolved, and the procedure can be repeated until an optimal solution is reached. The subproblem of identifying cuts is called *separation problem*, and it is of crucial importance to solve it efficiently, since many instances of it must usually be solved until the cutting plane approach terminates successfully.

Algorithm 1 The Generic Cutting-Plane Algorithm

Input: (ILP) $\max z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h$ s.t. $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$
repeat
 solve the LP-relaxation of **ILP**. Let \mathbf{x}^* be an optimal solution.
 if \mathbf{x}^* satisfies the integrality requirements **then**
 an optimal solution to **ILP** has been found. **stop**.
 else
 solve the separation problem, that is, try to find a valid inequality $\mathbf{w}^T \mathbf{x} \leq d$ such that $\mathbf{w}^T \mathbf{x}^* > d$.
 if such an inequality $\mathbf{w}^T \mathbf{x} \leq d$ cutting off \mathbf{x}^* was found **then**
 add the inequality to the system.
 else
 no optimal solution to **ILP** was found. **stop**.
 end if
 end if
until forever

In practice, it may take a long time for such a cutting plane approach to converge to the optimum, partly because it is often a hard subproblem to separate effective cuts. A further drawback of this technique is that no feasible integer solutions can be obtained until the optimal integer solution is reached, which implies that there is no feasible integer solution if the computations are stopped prematurely. The cutting plane method is therefore often combined with other methods, as we will see below.

Branch-and-bound methods

The basic structure of branch-and-bound is an *enumeration tree*. The *root* node of the tree corresponds to the original problem. As the algorithm progresses, the tree grows by a process called *branching*, which creates two or more child nodes of the parent node. Each of the problems at the child nodes is formed by adding constraints to the problem at the parent node. Typically, the new constraint is obtained by simply adding a bound on a single integer variable, where one child gets an upper bound of some integer d , and the other child gets a lower bound of $d + 1$. An essential requirement is that each feasible solution to the parent node problem is feasible to at least one of the child node problems.

Let $\mathbf{ILP}(0)$ be the original ILP problem and let $\mathbf{ILP}(k)$ be the problem at node k . The objective value of any feasible solution to $\mathbf{ILP}(k)$ provides a lower bound on the global optimal value. The feasible solution with the highest objective value found so far is called the *incumbent* solution and its objective value is denoted by z^{best} . Let \mathbf{x}^k be an optimal solution to the LP-relaxation of $\mathbf{ILP}(k)$ with objective value z^k . If \mathbf{x}^k satisfies the integrality constraints, then it is an optimal solution to $\mathbf{ILP}(k)$ and a feasible solution to $\mathbf{ILP}(0)$, and therefore we update z^{best} as $\max\{z^k, z^{best}\}$. Otherwise, there are two possibilities: if $z^k \leq z^{best}$, then an optimal solution to $\mathbf{ILP}(k)$ cannot improve on z^{best} , hence the subproblem $\mathbf{ILP}(k)$ is removed from consideration; on the other hand, if $z^k > z^{best}$, then $\mathbf{ILP}(k)$ requires further exploration, which is done by *branching*, i.e., by creating $q \geq 2$ new subproblems $\mathbf{ILP}(k(i))$, $i = 1, 2, \dots, q$, of $\mathbf{ILP}(k)$. Each feasible solution to $\mathbf{ILP}(k)$ must be feasible to at least one child and, conversely, each feasible solution to a child must be feasible to $\mathbf{ILP}(k)$. Moreover, the solution \mathbf{x}^k must not be feasible to any of the LP-relaxations of the children. A simple realization of these requirements is to select a variable x_j for which x_j^k is not integer and to create two subproblems; in one subproblem, we add the constraint $x_j \leq \lfloor x_j^k \rfloor$, which is the round down of x_j^k , and in the other $x_j \geq \lceil x_j^k \rceil$, which is the round up of x_j^k . The child nodes of node k corresponding to these subproblems are then added to the tree. The largest among all LP-relaxation values associated with the active subproblems provides a global upper bound on the optimal value. The algorithm terminates when the global upper bound and global lower bound (z^{best}) are equal.

Algorithm 2 The Branch-and-Bound Algorithm

Input: (\mathbf{ILP}) $\max z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h$ s.t. $\mathbf{Ax} \leq \mathbf{b}$, $\mathbf{x} \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$.

0. Initialize.

Create a list \mathbf{L} of active subproblems. Set $\mathbf{L} = \{\mathbf{ILP}(0)\}$, $z^{best} = -\infty$ and $\mathbf{x}^{best} = \emptyset$.

1. Terminate?

Is $\mathbf{L} = \emptyset$? If so, **return** \mathbf{x}^{best} is an optimal solution to \mathbf{ILP} .

2. Select.

Choose and delete a problem $\mathbf{ILP}(k)$ from \mathbf{L} .

3. Evaluate.

Solve the LP-relaxation $\mathbf{LP}(k)$ of $\mathbf{ILP}(k)$. If $\mathbf{LP}(k)$ is infeasible, **goto** Step 1, else let z^k be its objective function value and \mathbf{x}^k be its solution.

4. Prune.

If $z^k \leq z^{best}$, **goto** Step 1. If \mathbf{x}^k is not integer, **goto** Step 5, else let $z^{best} = z^k$, $\mathbf{x}^{best} = \mathbf{x}^k$. **Goto** Step 1.

5. Branch.

Divide the feasible domain S^k of $\mathbf{ILP}(k)$ into smaller sets $S^{k(i)}$ for $i = 1, \dots, q$, such that $\bigcup_{i=1}^q S^{k(i)} = S^k$, and add the subproblems $\mathbf{ILP}(k(i))$, $i = 1, \dots, q$, to \mathbf{L} . **Goto** Step 1.

This basic scheme does not specify the rule to follow for choosing a node from \mathbf{L} . A popular method to do this is to select the node with the highest value z^k . Such strategy is known as *best-bound search* (or *best-first search*). This node selection strategy focuses the search on decreasing the global upper bound, because the only way to decrease the global upper bound is to improve the LP-relaxation at a node with the highest LP-relaxation value. Another well-known method of selecting a node to explore is to always choose the most recently created node. This is known as *diving search* (or *depth-first search*). This node selection strategy focuses the search on increasing the global lower bound, because feasible solutions are typically found deep in the tree. In addition to a different focus, both methods also have different computational attributes. Diving search has low memory requirements, because only the sibling nodes on the path to the root of the tree have to be stored. Furthermore, the changes in the linear programs from one node to the next are minimal, a single bound of a variable changes, which allows warm-starts in the LP solves. Best-bound search, on the other hand, favors exploring nodes at the top of the tree as these are more likely to have high LP-relaxation values. This, however, can lead to large list of active subproblems. Furthermore, subsequent linear programs have little relation to each other leading to longer solution times.

We say that node k is *superfluous* if $z^k < z_{\text{ILP}}$. Best-bound search ensures that no superfluous nodes will be explored. On the other hand, diving search can lead to the exploration of many superfluous nodes that would have been fathomed, had we known a smaller z^{best} .

Most integer-programming solvers employ a hybrid of best-bound search and diving search, trying to benefit from the strengths of both, and switch regularly between the two strategies during the search. In the beginning the emphasis is usually more on diving, to find high quality solutions quickly, whereas in the later stages of the search, the emphasis is usually more on best-bound, to improve the global upper bound.

Combining branch-and-bound with cutting plane algorithms yields the highly effective class of *branch-and-cut* algorithms which are widely used in commercial ILP-solvers such as CPLEX and Gurobi [8]. Cuts are generated at the nodes of the branch-and-bound search tree to tighten the bounds of the LP-relaxations or to exclude infeasible solutions.

3 The PSA-pilp algorithm

In Section 2 we carry out a review of the main algorithms employed in the resolution of ILP problems. In all cases, we have seen that the strategy for finding the optimal solution consists of modifying the problem domain (having previously considered its relaxation) through the addition of new constraints. In the case of the cutting planes algorithms, the new inequalities are used to separate fractional solutions of the LP-relaxation and to keep the set of integer solutions of the original ILP problem. In the case of the branch-and-bound and related methods, the inequalities are used for partitioning the problem domain and eliminating fractional solutions of the LP-relaxation.

With a different approach, in this section we present the PSA-pilp algorithm, which does not alter the problem domain and, consequently, avoids the addition of new constraints to the original formulation.

3.1 Definitions and terminology

Let us begin this section by introducing the concepts of *projection*, *level* and *range* needed to describe the PSA-pilp algorithm. To this end, consider the two-variable PILP problem illustrated in figure 1 where: (i) it is supposed that the problem is in the form (2); (ii) the set of integer solutions is represented as black points on the (x_1, x_2) plane; and (iii) \mathbf{P}_j , $j = 1, 2$, denotes the *projection with respect to the variable x_j* , i.e., the shadow cast by $z(\mathbf{x})$ on the (x_j, z) plane.

From figure 1, it can be observed that the projection \mathbf{P}_j is defined on the interval $[l_j, u_j]$, where the endpoints of this interval clearly represent the minimum and maximum values of the variable x_j over the feasible domain of the LP-relaxation of the problem being solved. Thus, l_j and u_j can be formally defined as follows:

$$l_j = z_{\text{LP}}, \text{ with } (\text{LP}) \min z(\mathbf{x}) = x_j \text{ s.t. } \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^2;$$

$$u_j = z_{\text{LP}}, \text{ with } (\text{LP}) \max z(\mathbf{x}) = x_j \text{ s.t. } \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^2.$$

The projection \mathbf{P}_j can then be described as the two-dimensional convex set enclosed by the curves $P_j^{\text{low}}(x_j) : [l_j, u_j] \rightarrow \mathbb{R}$ and $P_j^{\text{up}}(x_j) : [l_j, u_j] \rightarrow \mathbb{R}$. The former function corresponds to the lower boundary of the set, which we will call the *lower projection*, and the latter corresponds to the upper boundary, which we will call the *upper projection*. It is straightforward to see that these curves can be calculated, for each fixed value $x_j = \lambda_j$, by solving two LP problems of one variable:

$$P_j^{\text{low}}(\lambda_j) = z_{\text{LP}}, \text{ with } (\text{LP}) \min z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h \text{ s.t. } x_j = \lambda_j, \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^2;$$

$$P_j^{\text{up}}(\lambda_j) = z_{\text{LP}}, \text{ with } (\text{LP}) \max z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h \text{ s.t. } x_j = \lambda_j, \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^2.$$

Consequently, for a two-variable problem, the projection of $z(\mathbf{x})$ onto the (x_j, z) plane can be defined as follows:

$$\mathbf{P}_j := \{ (x_j, z) \in \mathbb{R}^2 : x_j \in [l_j, u_j], P_j^{\text{low}}(x_j) \leq z \leq P_j^{\text{up}}(x_j) \}.$$

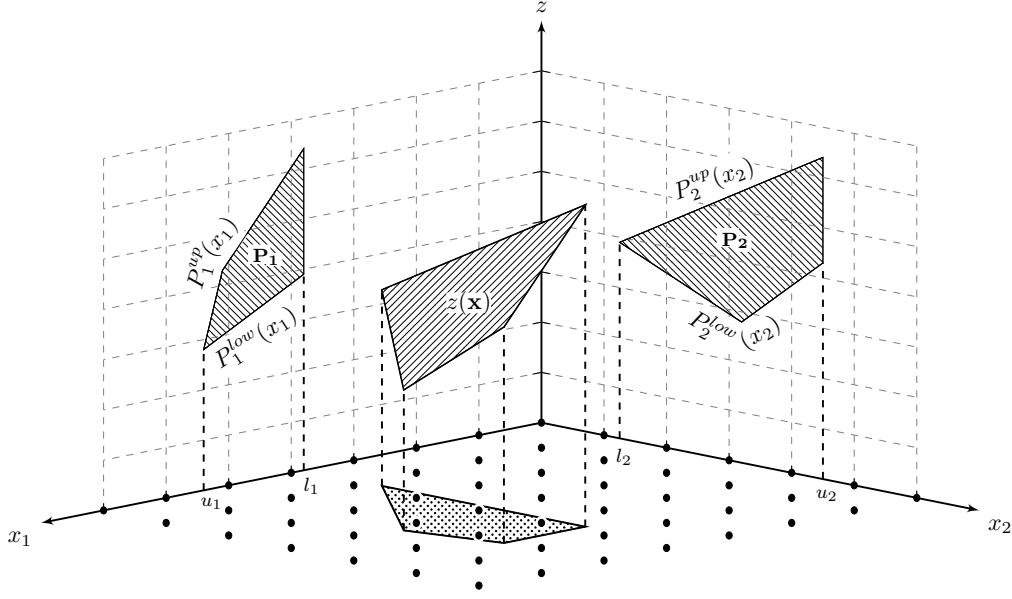


Fig. 1 orthogonal projections for a two-variable PILP problem

The definition of *projection* to be used in this paper is the natural extension of the model introduced above adapted to higher dimensions.

Definition 3.1 (Projection) Given a PILP problem, for $j = 1, \dots, n$ we define the projection of $z(\mathbf{x})$ onto the (x_j, z) plane, \mathbf{P}_j for short, as the two-dimensional convex set satisfying the following conditions.

$$\mathbf{P}_j := \{ (x_j, z) \in \mathbb{R}^2 : x_j \in [l_j, u_j], P_j^{\text{low}}(x_j) \leq z \leq P_j^{\text{up}}(x_j) \}, \quad (3)$$

where

$$l_j := z_{\text{LP}}, \text{ with } (\text{LP}) \min z(\mathbf{x}) = x_j \text{ s.t. } \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^n; \quad (4)$$

$$u_j := z_{\text{LP}}, \text{ with } (\text{LP}) \max z(\mathbf{x}) = x_j \text{ s.t. } \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^n; \quad (5)$$

and where the lower and upper projections, $P_j^{\text{low}}(x_j)$ and $P_j^{\text{up}}(x_j)$, can be determined, for each fixed value $x_j = \lambda_j \in [l_j, u_j]$, by solving two LP problems of $n - 1$ variables:

$$P_j^{\text{low}}(\lambda_j) = z_{\text{LP}}, \text{ with } (\text{LP}) \min z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h \text{ s.t. } x_j = \lambda_j, \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^n; \quad (6)$$

$$P_j^{\text{up}}(\lambda_j) = z_{\text{LP}}, \text{ with } (\text{LP}) \max z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h \text{ s.t. } x_j = \lambda_j, \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}_+^n. \quad (7)$$

Let us now introduce the concepts of *level* and *range* which will be used to interpret the information given by the projections.

Definition 3.2 (Level) Given a PILP problem, we will call level to each of the values that may be reached by the objective function $z(\mathbf{x})$. More precisely, we will call level to each of the elements of the $\text{cod}(z(\mathbf{x}))$ set.

Definition 3.3 (Range) Given a PILP problem, the set of integer values that can be assigned to the variable x_j , $j = 1, \dots, n$, when the projection \mathbf{P}_j is restricted to level z , will be called the range of x_j on level z . This set will be denoted by Range_j^z , and a more formal definition is given by:

$$\text{Range}_j^z = \{ r \in \mathbb{Z} : (r, z) \in \mathbf{P}_j \}. \quad (8)$$

To fix ideas, reconsider the projections \mathbf{P}_1 and \mathbf{P}_2 of the PILP problem shown in figure 1. Given that $\text{cod}(z(\mathbf{x})) \subseteq \mathbb{Z}$, it is straightforward to see that the set of values that may be reached by the objective function is given by: $\text{cod}(z(\mathbf{x})) = \{3, 4, 5, 6\}$. The figure presented below illustrates the two largest elements of this set along with the range of integer values that can be assigned to the variables x_1 and x_2 at each of those levels.

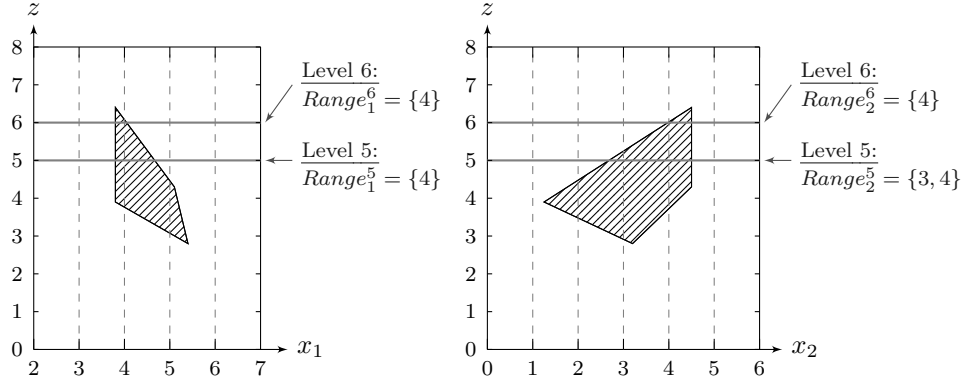


Fig. 2 projections \mathbf{P}_1 and \mathbf{P}_2 of the previous example crossed by levels 6 and 5

3.2 Motivation

Let us now explain the main steps involved in the PSA-*pilp* algorithm through the solution of the following instance of the classical *Unbounded Knapsack Problem* (UKP). This example covers all possibilities that may occur when applying PSAs for solving PILP problems.

$$\begin{aligned}
 (\mathbf{UKP}) \quad & \text{maximize } z(x_1, x_2, x_3) = 9x_1 + 3x_2 + 8x_3 \\
 & \text{subject to } 10x_1 + 5x_2 + 7x_3 \leq 12 \\
 & \mathbf{x} = (x_1, x_2, x_3) \in \mathbb{Z}_+^3
 \end{aligned}$$

As stated in [11], the LP-relaxation of every instance of the UKP can be trivially solved by comparing the quotients $\frac{c_j}{a_{1j}}$ corresponding to each variable x_j . For this reason, the projections \mathbf{P}_j in this example can be exactly computed by simply applying the expressions (3) to (7) to the proposed formulation. Thus, the family of projections of **UKP** turns out to be (see figure 3):

$$\begin{aligned}
 \mathbf{P}_1 &= \left\{ (x_1, z) \in \mathbb{R}^2 : x_1 \in \left[0, \frac{12}{10}\right], 9x_1 \leq z \leq -\frac{17}{7}x_1 + \frac{96}{7} \right\}, \\
 \mathbf{P}_2 &= \left\{ (x_2, z) \in \mathbb{R}^2 : x_2 \in \left[0, \frac{12}{5}\right], 3x_2 \leq z \leq -\frac{19}{7}x_2 + \frac{96}{7} \right\}, \\
 \mathbf{P}_3 &= \left\{ (x_3, z) \in \mathbb{R}^2 : x_3 \in \left[0, \frac{12}{7}\right], 8x_3 \leq z \leq \frac{17}{10}x_3 + \frac{108}{10} \right\}.
 \end{aligned}$$

These projections make it possible to decompose the original problem into single-variable subproblems, and thus they allow us to study the behaviour of the objective function from each variable's point of view independently. In particular, every time a specific value of the objective function is observed (think of a horizontal line across \mathbf{P}_1 , \mathbf{P}_2 and \mathbf{P}_3), the information given by the projections can be used to restrict the *range* of integer values that can be assigned to each variable x_j . As a result, candidate solutions capable of reaching the selected z -value can be generated by combining the allowed values of each of the \mathbf{x} -coordinates.

Given that the set of all possible optimal objective values of **UKP** is finite, namely $\text{cod}(z(\mathbf{x})) = \{0, 1, \dots, 12, 13\}$, it becomes natural to address the solution of **UKP** by studying the candidate solutions produced by applying the observation made above to each of the elements of the $\text{cod}(z(\mathbf{x}))$ set. Furthermore, because we are maximizing, we can conduct the search process for the optimal solution by considering, one by one in decreasing order of value, each of the elements of the $\text{cod}(z(\mathbf{x}))$ set. Then it is easy to see that, if a *feasible* candidate solution $\bar{\mathbf{x}}$ satisfying the condition $z(\bar{\mathbf{x}}) = z_i$ is found when level $z_i \in \text{cod}(z(\mathbf{x}))$ is being observed, this automatically implies that $\bar{\mathbf{x}}$ is a global optimum to the proposed problem.

To show more clearly what we are saying, reconsider the projections \mathbf{P}_1 , \mathbf{P}_2 and \mathbf{P}_3 of the problem at hand together with the three largest elements of the $\text{cod}(z(\mathbf{x}))$ set. Figure 3 illustrates this situation along with the range of integer values that can be assigned to the variables x_1 , x_2 and x_3 at each of those levels.

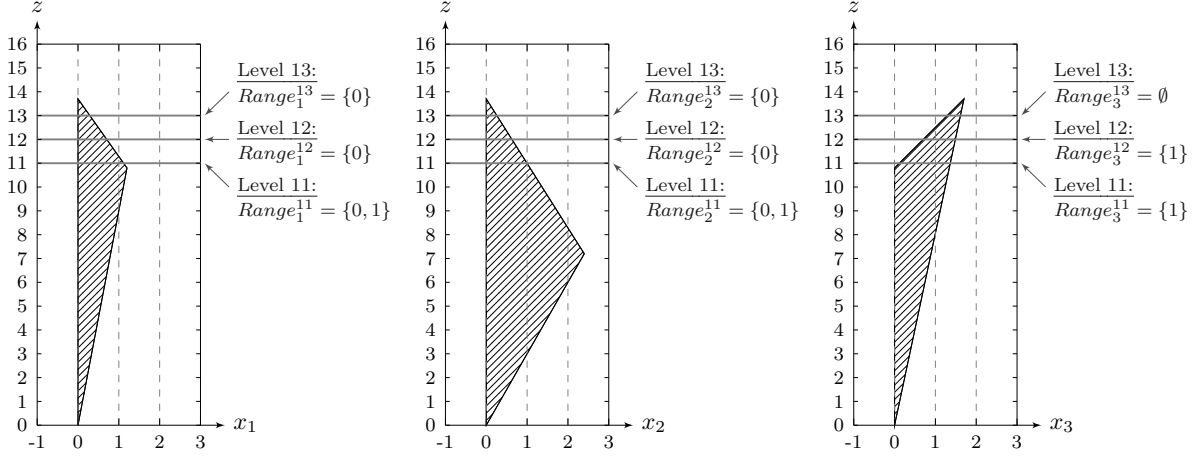


Fig. 3 projections \mathbf{P}_1 , \mathbf{P}_2 and \mathbf{P}_3 of UKP crossed by levels 13, 12, and 11

Level 13. Based on the values contained in the sets Range_j^{13} , $j = 1, 2, 3$, it can be inferred that there is no candidate solution capable of reaching level 13, i.e., we can conclude that $z_{\text{UKP}} < 13$. The level 13 is then discarded from the list of possible optimal objective values of **UKP**, and the search process is continued at level 12.

Level 12. From the information given by the sets Range_j^{12} , $j = 1, 2, 3$, it can be inferred that $\bar{\mathbf{x}} = (0, 0, 1)$ is the *unique* candidate solution capable of reaching level 12. Then, to determine whether 12 is the optimal objective value of the problem and $\bar{\mathbf{x}} = (0, 0, 1)$ is the associated optimal solution, we simply check the following two conditions (from now on the *stopping criterion*) on our candidate solution: (i) is $\bar{\mathbf{x}}$ feasible? (ii) does $z(\bar{\mathbf{x}}) = 12$? If the answer to both questions is affirmative, clearly $\bar{\mathbf{x}} = (0, 0, 1)$ is an optimal solution to the proposed problem and 12 is the optimal objective value; otherwise, level 12 is discarded from the list of possible optimal objective values of **UKP**, and the search process is continued at level 11.

By a simple calculation, it is easy to see that $\bar{\mathbf{x}} = (0, 0, 1)$ is a feasible solution to **UKP**, however, it yields an objective value of 8. Hence, given that $\bar{\mathbf{x}} = (0, 0, 1)$ is the unique candidate solution arising from this value of the maximand, it can be concluded that: $z_{\text{UKP}} < 12$ and $z_{\text{UKP}} \geq 8$. Before considering the next level and continuing with the search process, it is necessary to introduce two new variables in order to keep the former information: $\bar{\mathbf{x}}^{\text{best}} := (0, 0, 1)$ (*incumbent solution*); $z^{\text{best}} := 8$ (*lower bound*).

Level 11. From the sets Range_j^{11} , $j = 1, 2, 3$, it can be inferred that the points $\bar{\mathbf{x}} = (0, 0, 1)$, $\bar{\mathbf{x}} = (0, 1, 1)$, $\bar{\mathbf{x}} = (1, 0, 1)$ and $\bar{\mathbf{x}} = (1, 1, 1)$ are the *only* four candidates for level 11. We can now proceed in two different ways in order to determine whether some of these candidates is, in fact, an optimal solution to the proposed problem. The first alternative is to repeat what was done at the previous level, i.e., to simply check the stopping criterion on each of the four candidate solutions. If we are thinking of extending the procedure to higher dimensions, this approach is clearly inefficient due to the exponential growth in the number of candidates. The second alternative, which is the one we are going to use, is to try to extract a little more of the information contained in \mathbf{P}_1 , \mathbf{P}_2 and \mathbf{P}_3 in order to reduce the number of candidate solutions arising from the level being scanned.

With this latter goal in mind, we begin by observing that the condition $|\text{Range}_3^{11}| = 1$ implies that all possible candidate solutions for the current level must be in the form $\bar{\mathbf{x}} = (?, ?, 1)$ (such a point will be called a *partial candidate solution* to **UKP**). Then, the restriction $x_3 = 1$ can be imposed on the original formulation, thus obtaining a new problem of a smaller dimension. Hereafter, the resulting problem will be called the *reduced problem*, and we will denote it by $\text{UKP}|_{\bar{\mathbf{x}}}$. In our case, the reduced problem turns out to be:

$$(\text{UKP}|_{\bar{\mathbf{x}}}) \max z(x_1, x_2, 1) = 9x_1 + 3x_2 + 8 \text{ s.t. } 10x_1 + 5x_2 \leq 5, (x_1, x_2) \in \mathbb{Z}_+^2.$$

Now, the general procedure can be applied to the reduced problem: recalculate the projections \mathbf{P}_1 and \mathbf{P}_2 , and re-examine level 11 in order to determine the new sets Range_1^{11} and Range_2^{11} .

Before recalculating \mathbf{P}_1 and \mathbf{P}_2 explicitly and carrying on with the example, let us open a parenthesis here to enumerate the 4 alternatives that may hold depending on the cardinality of the new sets $Range_1^{11}$ and $Range_2^{11}$. We will also explain how to proceed in each situation. For convenience in the exposition, the set consisting of the variables that have not yet been fixed will be called *active variables* (\mathbf{AV}). In our case, $\mathbf{AV} = \{x_1, x_2\}$.

1. **if $|Range_j^{11}| = 1$ for all j such that $\mathbf{x}_j \in \mathbf{AV}$.** This means that there exist values $a, b \in \mathbb{Z}$ such that $Range_1^{11} = \{a\}$ and $Range_2^{11} = \{b\}$. Then, we can assert that, if there existed a feasible solution for this value of the maximand, it should be in the form $\bar{\mathbf{x}} = (a, b, 1)$. The search process finishes if the resulting point satisfies the stopping criterion. Otherwise, given that $\bar{\mathbf{x}} = (a, b, 1)$ is the only candidate solution arising from this level, we can conclude that $z_{\mathbf{UKP}} < 11$. In the latter case, before proceeding to the next level and continuing with the search process, we first check whether the variables $\bar{\mathbf{x}}^{best}$ and z^{best} can be updated.
2. **if $|Range_j^{11}| = 0$ for at least one j such that $\mathbf{x}_j \in \mathbf{AV}$.** In this case, there is no integer value that can be assigned to, at least, one of the non-fixed coordinates of $\bar{\mathbf{x}} = (?, ?, 1)$. Therefore, we can conclude that $z_{\mathbf{UKP}} < 11$. Then, the original problem is reconsidered and the search process is restarted at level 10.
3. **if $|Range_j^{11}| = 1$ for at least one j such that $\mathbf{x}_j \in \mathbf{AV}$ (but not all).** Without loss of generality, let us suppose that $|Range_1^{11}| = 1$, i.e., there exists a value $a \in \mathbb{Z}$ such that $Range_1^{11} = \{a\}$. Then, the partial candidate solution, the set of active variables, and the reduced problem can be updated as follows:

$$\bar{\mathbf{x}} = (a, ?, 1), \mathbf{AV} = \{x_2\}, \text{ and } (\mathbf{UKP}|_{\bar{\mathbf{x}}}) \max z(a, x_2, 1) = 3x_2 + 8 + 9a \text{ s.t. } 5x_2 \leq 5 - 10a, x_2 \in \mathbb{Z}_+.$$

In this way, the original problem is further reduced in size, and the process can be continued (at the current level) by recalculating the projection \mathbf{P}_2 of $\mathbf{UKP}|_{\bar{\mathbf{x}}}$, and by performing the same four-step analysis that is being used here.

4. **if $|Range_j^{11}| > 1$ for all j such that $\mathbf{x}_j \in \mathbf{AV}$.** In this case, we proceed in the following manner. Firstly, we choose one of the active variables of the problem using some criterion, say x_s , and create new partial candidate solutions by assigning the r^{th} value contained in the set $Range_s^{11}$, $1 \leq r \leq |Range_s^{11}|$, to the s^{th} component of $\bar{\mathbf{x}}$. By abuse of notation, we will also write $\bar{\mathbf{x}}$ to denote the new partial candidate solutions created in this manner. Secondly, we add all the partial candidate solutions constructed in the previous step to the set of partial candidate solutions to be analysed (\mathbf{L}). Thirdly, using some criterion, we extract one of the partial candidate solutions added to \mathbf{L} , say $\bar{\mathbf{x}}$, and calculate the reduced problem associated to it ($\mathbf{UKP}|_{\bar{\mathbf{x}}}$) and redefine \mathbf{AV} as the set of non-fixed components of $\bar{\mathbf{x}}$. Finally, the search process is continued by recalculating the sets \mathbf{P}_j and $Range_j^{11}$ of $\mathbf{UKP}|_{\bar{\mathbf{x}}}$ (for the variable x_j that has not yet been fixed), and by performing the same four-step analysis that is being used here. Note that, if the current problem does not produce any optimal solution, it is necessary to analyse the solution space generated by the remaining partial candidate solutions contained in \mathbf{L} before concluding that 11 is not the optimal level of $z(\mathbf{x})$. If any optimal solution is reached, the procedure terminates; otherwise, the original problem is reconsidered and the search process is restarted at level 10.

Having established the 4 alternatives that may hold depending on the cardinality of the new sets $Range_1^{11}$ and $Range_2^{11}$, let us now come back to the example. In our case, the projections \mathbf{P}_1 and \mathbf{P}_2 of the reduced problem turns out to be (see figure 4):

$$\mathbf{P}_1 = \left\{ (x_1, z) \in \mathbb{R}^2 : x_1 \in \left[0, \frac{1}{2}\right], 9x_1 + 8 \leq z \leq \frac{15}{5}x_1 + \frac{55}{5} \right\},$$

$$\mathbf{P}_2 = \left\{ (x_2, z) \in \mathbb{R}^2 : x_2 \in [0, 1], 3x_2 + 8 \leq z \leq -\frac{15}{10}x_2 + \frac{125}{10} \right\}.$$

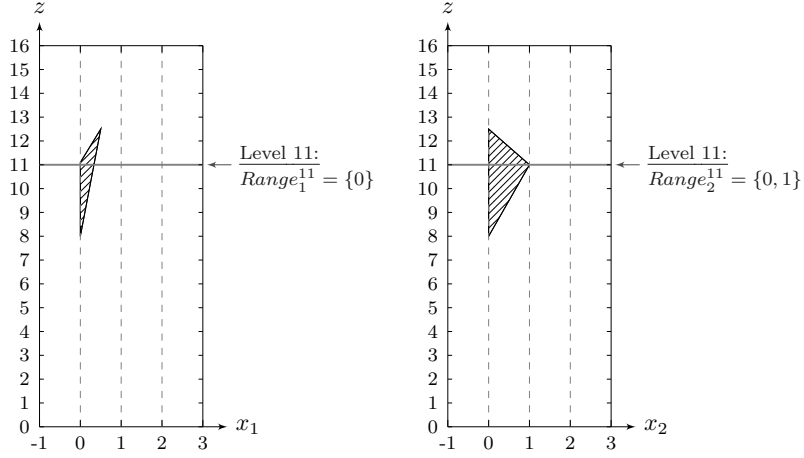


Fig. 4 projections \mathbf{P}_1 and \mathbf{P}_2 of $\mathbf{UKP}|_{(\cdot, \cdot, 1)}$ crossed by level 11

From figure 4, it can be observed that the range of integer values that can be assigned to each of the remaining active variables is given by: $\text{Range}_1^{11} = \{0\}$ and $\text{Range}_2^{11} = \{0, 1\}$ (case 3). Therefore, the partial candidate solution, the set of active variables, and the reduced problem can be updated as follows:

$$\bar{\mathbf{x}} = (0, ?, 1), \mathbf{AV} = \{x_2\}, \text{ and } (\mathbf{UKP})_{|\bar{\mathbf{x}})} \max z(0, x_2, 1) = 3x_2 + 8 \text{ s.t. } 5x_2 \leq 5, x_2 \in \mathbb{Z}_+.$$

The projection \mathbf{P}_2 of $\mathbf{UKP}_{|\bar{\mathbf{x}})}$ is then recalculated in an attempt to obtain tighter bounds for the set Range_2^{11} (see figure 5):

$$\mathbf{P}_2 = \left\{ (x_2, z) \in \mathbb{R}^2 : x_2 \in [0, 1], 3x_2 + 8 \leq z \leq 3x_2 + 8 \right\}.$$

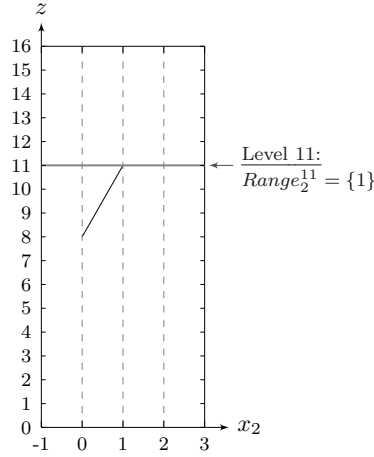


Fig. 5 projection \mathbf{P}_2 of $\mathbf{UKP}|_{(0, \cdot, 1)}$ crossed by level 11

From figure 5 it follows that: $\text{Range}_2^{11} = \{1\}$ (case 1). This means that $\bar{\mathbf{x}} = (0, 1, 1)$ is the unique candidate solution capable of reaching level 11. Then, given that $\bar{\mathbf{x}}$ satisfies the stopping criterion, we can conclude that it is an optimal solution to \mathbf{UKP} . \triangleleft

The algorithm to be described in this paper is the generalization of the above procedure adapted to higher dimensions. The following outline summarizes how the proposed algorithm works. Given a PILP problem, the

PSA-pilp algorithm starts by calculating the orthogonal projection associated to each variable x_j , and by identifying the set of all possible optimal objective values, say $\text{cod}(z(\mathbf{x})) = \{z_1, \dots, z_k\}$. Then, it begins the search for the optimal solution by considering, one by one in decreasing order of value, each of the elements of the $\text{cod}(z(\mathbf{x}))$ set. Every time a new level $z_i \in \text{cod}(z(\mathbf{x}))$ is selected, the algorithm utilizes the information contained in the sets $\text{Range}_j^{z_i}$ to fix the value of some of the variables, and thus to reduce the size of the original problem. The procedure is then continued by recalculating the sets \mathbf{P}_j and $\text{Range}_j^{z_i}$ of the reduced problem for the variables that have not yet been fixed (*active variables*). A number of candidate solutions is constructed for each considered level z_i by applying this argument systematically.

The search process ends (stopping criterion) when a *feasible candidate solution* $\bar{\mathbf{x}}$ satisfying the condition $z(\bar{\mathbf{x}}) = z_i$ is found when the algorithm is scanning level z_i . Then, it can be concluded that $\bar{\mathbf{x}}$ is a global optimum to the proposed PILP problem.

As can be seen in this outline, and also in the previous example, the proposed algorithm differs from the state-of-the-art techniques in three aspects: (i) it guides the search for the optimal solution by generating candidate solutions tailored to specific values of the objective function; (ii) it systematically reduces the number of variables in the original problem for each considered level; and (iii) it does not add any additional constraint to the initial formulation. Concerning the second point, it is worth noting that, while in the case of branch-and-bound-based algorithms the number of variables that can be fixed in each iteration of the procedure (for each node in the search tree) oscillates between 0 and 1, in the case of the PSA-pilp algorithm this figure ranges between 1 and $|\mathbf{AV}|$. Furthermore, as we will see later on in Section 5, the computational experiments performed on instances of the 0-1MKP reveal that the percentage of variables that are fixed to their optimal value in the *first iteration* of the PSA-pilp algorithm at the optimal level, rise to more than 97% of the total variables.

3.3 Practical aspects

To conclude this section, let us give some precisions about how to calculate projections in the case of general PILP problems. This is motivated by the fact that, unlike what happened in Section 3.2 for the UKP, in the case of general PILP problems it is usually too expensive—or even impossible—to derive explicit formulas for $P_j^{\text{up}}(x_j)$ and $P_j^{\text{low}}(x_j)$ for all x_j in the domain of the definition of \mathbf{P}_j . It then becomes necessary to identify which part of the information provided by the projections is dispensable and which part is strictly necessary for executing the PSA-pilp algorithm.

It is easy to see that the only information that is absolutely necessary for executing the PSA-pilp algorithm is that given by the points $(e_j, P_j^{\text{up}}(e_j))$ and $(e_j, P_j^{\text{low}}(e_j))$, where e_j takes on all possible *integer* values in the domain of the definition of \mathbf{P}_j . From a theoretical point of view, this observation makes it possible to compute the set of projections for every instance of a PILP problem in a finite number of steps. In practice, however, it may take a long time for the PSA-pilp algorithm to converge to the optimum if the coefficients $P_j^{\text{up}}(e_j)$ and $P_j^{\text{low}}(e_j)$ are calculated exactly. It is then natural to, in addition to the previous simplification, approximate some of these values in order to reduce the number of operations even further.

To fix ideas, the following outline details the steps of the procedure suggested above applied to the computation of the set of upper projections, $\{P_j^{\text{up}}(e_j) \mid e_j \text{ integer in the domain of the definition of } \mathbf{P}_j\}$, for the subclass of PILP problems in which all variables are restricted to be 0 or 1. This type of problems is known as *binary integer linear programming* (BILP). A similar approach can be applied to determine the lower projections of a BILP problem as well as to calculate the upper and lower projections for more complex PILP problems.

- *Phase 1.* The integer requirements of the original BILP problem are relaxed and the associated maximization LP program is solved by using the Simplex method. Let \mathbf{x}^* denote the optimal solution to the LP-relaxation, and let z_{LP} denote its respective optimal objective value. It is easy to see that, if the j^{th} component of \mathbf{x}^* yields an integer value $e_j \in \{0, 1\}$, this automatically implies that $P_j^{\text{up}}(e_j) = z_{\text{LP}}$. In other words, assuming that the problem we are trying to solve had n variables, this first operation would allow us to calculate, in the best-case scenario, up to $n - 1$ of the total $2n$ coefficients $P_j^{\text{up}}(e_j)$, $e_j \in \{0, 1\}$.
- *Phase 2.* For each of the remaining values, $P_j^{\text{up}}(e_j)$, that were not able to be computed in the previous phase, the additional constraint $x_j \leq 0$ (if $e_j = 0$) or $x_j \geq 1$ (if $e_j = 1$) is added to the bottom of the optimal Simplex tableau obtained in the previous step, and the dual Simplex algorithm is then used to restore primal feasibility and to compute an upper bound for $P_j^{\text{up}}(e_j)$.

4 Scheme and correctness of the PSA-*pilp* algorithm

In this section, we present the scheme of the algorithm. In order to do that, we assume that there exists a procedure that permits to compute the set of projections for every instance of a PILP problem in a finite number of operations. The same assumption will be made on Section 4.2 to prove the finiteness and the correctness of the algorithm.

4.1 Scheme of the algorithm

As we mentioned before, the strategy of the PSA-*pilp* algorithm is to sweep across the set of all possible optimal objective values of the problem, say $cod(z(\mathbf{x})) = \{z_1, \dots, z_k\}$, and to use the information given by the sets $Range_j^{z_j}$, j such that $x_j \in \mathbf{AV}$, to generate a finite number of candidate solutions tailored to each of the selected z -values. The search process finishes when a candidate solution which meets the stopping condition is found.

In order to clarify the exposition of the algorithm, we will divide the procedure into two parts, thus introducing a slight modification in comparison to the example presented in Section 3.2. On the one hand, we will introduce the `Inspect_Level` algorithm, which is the responsible for generating the whole set of candidate solutions associated to a given level. On the other hand, we will present the `Main` algorithm, which is the responsible for performing the parallel shifts in the functional value in the direction of a reduction of the maximand, and for checking the stopping criterion on the set of candidate solutions provided by the `Inspect_Level` algorithm. The scheme of the algorithms is as follows:

Algorithm 3 The Main Algorithm

Input: (**PILP**) $\max z(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + h$ s.t. $\mathbf{Ax} \leq \mathbf{b}$, $\mathbf{x} \in \mathbb{Z}_+^n$

Assumption: $\mathbf{c} \in \mathbb{Z}^n, h \in \mathbb{Z}$

Output: optimal solution to **PILP**, or detects infeasibility

Variables:

$\bar{\mathbf{x}} = (\bar{x}_1, \dots, \bar{x}_n) \in \mathbb{Z}^n$ (candidate solution)

$cod(z(\mathbf{x})) \subseteq \mathbb{Z}$ (codomain of $z(\mathbf{x})$ over the feasible domain of **PILP**)

$z \in \mathbb{Z}$ (level being scanned)

$z^{best} \in \mathbb{Z}$ (lower bound)

$\bar{\mathbf{x}}^{best} \in \mathbb{Z}^n$ (incumbent solution)

$\mathbf{CS}_z^{\mathbf{PILP}}$ (set of candidate solutions to **PILP** produced by the `Inspect_Level` algorithm at level z)

\mathbf{P}_j (projection produced by $z(\mathbf{x})$ onto the (x_j, z) plane)

0. Initialize.

compute \mathbf{P}_j for $j = 1, \dots, n$

compute $cod(z(\mathbf{x}))$

set z to the largest element in $cod(z(\mathbf{x}))$

set z^{best} to the smallest element in $cod(z(\mathbf{x}))$

1. Loop.

while $z > z^{best}$ **or** ($z == z^{best}$ **and** $\bar{\mathbf{x}}^{best} == \text{NULL}$) **do**

1.1. Inspection.

set $\mathbf{CS}_z^{\mathbf{PILP}} = \text{Inspect_Level}(\mathbf{PILP}, z, \{\mathbf{P}_j\}_{j=1, \dots, n})$

1.2. Check.

for all $\bar{\mathbf{x}} \in \mathbf{CS}_z^{\mathbf{PILP}}$ **do**

if $\bar{\mathbf{x}}$ is a feasible solution to **PILP** **and** $z(\bar{\mathbf{x}}) == z$

return $\bar{\mathbf{x}}$ is an optimal solution to **PILP**

else if $\bar{\mathbf{x}}$ is feasible **and** $z(\bar{\mathbf{x}}) > z^{best}$

```

                                set  $z^{best} = z(\bar{\mathbf{x}})$ 
                                set  $\bar{\mathbf{x}}^{best} = \bar{\mathbf{x}}$ 
                            end if
                        end for

1.3. Next level.
        set  $z = z - 1$ 
    end while

2. Output.
    if  $\bar{\mathbf{x}}^{best} \neq \text{NULL}$ 
        return  $\bar{\mathbf{x}}^{best}$  is an optimal solution to PILP
    else
        return PILP is infeasible
    end if

```

Algorithm 4 The Inspect_Level Algorithm

Input: **PILP** problem, z (level to be scanned), $\{\mathbf{P}_j\}_{j=1,\dots,n}$ (set of projections associated to **PILP**)

Output: $\text{CS}_z^{\text{PILP}}$ (set of candidate solutions to **PILP** arising from level z)

Variables:

$\bar{\mathbf{x}} = (\bar{x}_1, \dots, \bar{x}_n) \in \mathbb{Z}^n$ (partial candidate solution)

Prob (problem being analysed)

AV (set of variables that have not yet been fixed)

$\{\mathbf{P}_j\}_{j:x_j \in \text{AV}}$ (set of projections associated to **Prob**)

L (set of partial candidate solutions to be analysed)

$\text{CS}_z^{\text{PILP}}$ (set of candidate solutions to **PILP** arising from level z)

0. Initialize.

```

set Prob = PILP
set AV =  $\{x_1, \dots, x_n\}$ 
set  $\{\mathbf{P}_j\}_{j:x_j \in \text{AV}} = \{\mathbf{P}_j\}_{j=1,\dots,n}$ 
set  $\text{CS}_z^{\text{PILP}} = \{\}$ 
set  $\bar{x}_j = \text{NULL}$  for all  $j = 1, \dots, n$ 
set L =  $\{\}$ 

```

1. Inspection.

```

compute  $\text{Range}_j^z$  for all  $j$  such that  $x_j \in \text{AV}$ 
if  $|\text{Range}_j^z| > 0$  for all  $j$  such that  $x_j \in \text{AV}$ 
    if  $\exists j$  such that  $x_j \in \text{AV}$  and  $|\text{Range}_j^z| == 1$ 
        for all  $j$  such that  $x_j \in \text{AV}$  and  $|\text{Range}_j^z| == 1$  do      /*  $\text{Range}_j^z = \{r_j\}$  */
            set  $\bar{x}_j = r_j$ 
            set AV = AV -  $\{x_j\}$ 
        end for
        if AV ==  $\emptyset$ 
            set  $\text{CS}_z^{\text{PILP}} = \text{CS}_z^{\text{PILP}} \cup \{\bar{\mathbf{x}}\}$ 
        else
            set Prob = PILP $_{|\bar{\mathbf{x}}}$ 
            compute the set of projections associated to Prob:  $\{\mathbf{P}_j\}_{j:x_j \in \text{AV}}$ 

```

```

        goto Step 1
    end if
else
    /*|Rangezj| > 1 ∀ j / xj ∈ AV*/
    choose j such that xj ∈ AV using some criterion    /*Rangezj = {r1, ..., r|Rangezj|}*/
    if |AV| == 1
        for i = 1 to |Rangezj| do
            set  $\bar{x}_j = r_i$ 
            set  $\text{CS}_z^{\text{PILP}} = \text{CS}_z^{\text{PILP}} \cup \{\bar{\mathbf{x}}\}$ 
        end for
    else
        for i = 1 to |Rangezj| do
            set  $\bar{x}_j = r_i$ 
            set  $\bar{\mathbf{L}} = \mathbf{L} \cup \{\bar{\mathbf{x}}\}$ 
        end for
    end if
end if
end if
end if

2. Update.
    if  $\mathbf{L} \neq \emptyset$ 
        choose  $\bar{\mathbf{x}} \in \mathbf{L}$  using some criterion
        set  $\mathbf{L} = \mathbf{L} - \{\bar{\mathbf{x}}\}$ 
        set AV = non-fixed components of  $\bar{\mathbf{x}}$ 
        set  $\mathbf{Prob} = \mathbf{PILP}|_{\bar{\mathbf{x}}}$ 
        compute the set of projections associated to  $\mathbf{Prob}$ :  $\{\mathbf{P}_j\}_{j: x_j \in \text{AV}}$ 
        goto Step 1
    else
        return  $\text{CS}_z^{\text{PILP}}$ 
    end if

```

4.2 Correctness of the PSA-pilp algorithm

This section is intended to prove that the algorithm finds an optimal solution, or detects infeasibility, in a finite number of iterations. Before we come to the theorem we will enunciate two lemmas.

Lemma 4.1 *Let $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_n)$ be a feasible solution to \mathbf{PILP} (2) such that $z(\tilde{\mathbf{x}}) = \tilde{z}$. Then, $\tilde{x}_j \in \text{Range}_{\tilde{z}}^j$ for all $j = 1, \dots, n$.*

Proof The result follows from the definitions (8), (3), (6) and (7), and from the fact that $\tilde{\mathbf{x}}$ is a feasible solution to \mathbf{PILP} . \square

Note that, when the projections are restricted to the optimal level of the problem, say $z_{\mathbf{PILP}}$, lemma 4.1 asserts that every optimal solution to \mathbf{PILP} can be reconstructed from the information provided by the sets $\text{Range}_{z_{\mathbf{PILP}}}^j$, $j = 1, \dots, n$.

Lemma 4.2 *Let $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ be an optimal solution to \mathbf{PILP} (2), and let $\bar{\mathbf{x}}$ be the partial candidate solution defined by $\bar{\mathbf{x}} = (?, \dots, ?, x_k^*, \dots, x_n^*)$, $k > 1$. Then, \mathbf{x}^* is optimal to $\mathbf{PILP}|_{\bar{\mathbf{x}}}$. Furthermore, the problems \mathbf{PILP} and $\mathbf{PILP}|_{\bar{\mathbf{x}}}$ have both the same optimal objective value.*

Proof The results follow from the fact that \mathbf{x}^* is feasible for both \mathbf{PILP} and $\mathbf{PILP}|_{\bar{\mathbf{x}}}$. \square

Theorem 4.3 *The PSA-pilp algorithm converges to an optimal solution, or detects infeasibility, in a finite number of steps.*

Proof The finiteness of the algorithm follows from the fact that there is always a finite number of levels and a finite number of candidate solutions to be analysed. For this reason, the algorithm always stops after a finite number of iterations. In the case that the PILP problem is infeasible, the algorithm terminates returning this condition.

To prove the correctness of the algorithm we need only to show that, if $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ is an optimal solution to **PILP** (2) and z_{PILP} is the optimal objective value of the problem, then

$$\mathbf{x}^* \in \mathbf{CS}_{z_{\text{PILP}}}^{\text{PILP}} = \text{Inspect_Level}(\mathbf{PILP}, z_{\text{PILP}}, \{\mathbf{P}_j\}_{j=1, \dots, n}).$$

We are going to prove this property by induction on the number of variables.

For one-variable PILP problems the situation is as follows:

$$\begin{aligned} (\mathbf{PILP}) \quad & \text{maximize } z(x_1) = c_1 x_1 + h \\ & \text{subject to } a_{11} x_1 \leq b_1 \\ & \vdots \\ & a_{m1} x_1 \leq b_m \\ & x_1 \in \mathbb{Z}_+ \end{aligned}$$

We assume, without loss of generality, that $c_1 > 0$ and $h = 0$. Let $[l_1, u_1]$, $l_1, u_1 \in \mathbb{R}$, be the feasible domain of the LP-relaxation of **PILP**. Then, the optimal solution to **PILP** is reached at $\mathbf{x}^* = (\lfloor u_1 \rfloor)$ yielding an objective value of $z_{\text{PILP}} = c_1 \lfloor u_1 \rfloor$. By applying the **Inspect_Level** algorithm to **PILP** restricted to level z_{PILP} , it is easy to see that $\text{Range}_1^{z_{\text{PILP}}} = \{\lfloor u_1 \rfloor\}$. This implies $\mathbf{CS}_{z_{\text{PILP}}}^{\text{PILP}} = \{\bar{\mathbf{x}} = (\lfloor u_1 \rfloor)\}$. Then, the theorem is true for every instance of a PILP problem with one variable.

Inductive step. Suppose that the result is valid for every PILP problem with k variables, $k < n$. Let us now demonstrate that the property is also valid for every PILP problem with n variables. Let **PILP** be a PILP problem with n variables satisfying (2), and let $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ be an optimal solution to **PILP**. From lemma 4.1, it follows that $x_j^* \in \text{Range}_j^{z_{\text{PILP}}} \forall j \in \{1, \dots, n\}$. Then, by applying the **Inspect_Level** algorithm to **PILP** restricted to level z_{PILP} , only one of the following alternatives holds:

1. $\exists j \in \{1, \dots, n\} \mid |\text{Range}_j^{z_{\text{PILP}}}| = 1$. Let us suppose, without loss of generality, that $|\text{Range}_j^{z_{\text{PILP}}}| = 1 \forall j \in \{k, \dots, n\}$ for some $k \geq 1$. That is, $\text{Range}_j^{z_{\text{PILP}}} = \{x_j^*\} \forall j \in \{k, \dots, n\}$.
 - (a) If $k = 1$, then $\mathbf{x}^* \in \mathbf{CS}_{z_{\text{PILP}}}^{\text{PILP}}$.
 - (b) If $k > 1$, the **Inspect_Level** algorithm updates the partial candidate solution, the set of active variables, and the reduced problem in the following manner:

$$\bar{\mathbf{x}} = (?, \dots, ?, x_k^*, \dots, x_n^*), \mathbf{AV} = \{x_1, \dots, x_{k-1}\}, \text{ and } \mathbf{Prob} = \mathbf{PILP}|_{\bar{\mathbf{x}}}.$$

It then recalculates $\{\mathbf{P}_j\}_{j: x_j \in \mathbf{AV}}$, and *restarts the process from step 1 until the algorithm ends*. We now observe that this last operation is equivalent to apply

$$\mathbf{CS}_{z_{\text{PILP}}}^{\text{Prob}} = \text{Inspect_Level}(\mathbf{Prob}, z_{\text{PILP}}, \{\mathbf{P}_j\}_{j: x_j \in \mathbf{AV}}),$$

and then to extend the set of candidate solutions produced by the **Inspect_Level** algorithm to a set of candidate solutions valid for **PILP**. This operation is performed by setting $\bar{x}_j = x_j^*$ ($j = k, \dots, n$) for all $\bar{\mathbf{x}} \in \mathbf{CS}_{z_{\text{PILP}}}^{\text{Prob}}$.

To conclude, to prove that $\mathbf{x}^* \in \mathbf{CS}_{z_{\text{PILP}}}^{\text{PILP}}$, it suffices to show that $(x_1^*, \dots, x_{k-1}^*) \in \mathbf{CS}_{z_{\text{PILP}}}^{\text{Prob}}$. This result follows from lemma 4.2 and the induction hypothesis.

2. $|\text{Range}_j^{z_{\text{PILP}}}| > 1 \forall j \in \{1, \dots, n\}$. Without loss of generality, let us consider that index n is chosen. For each value $r_i \in \text{Range}_n^{z_{\text{PILP}}} = \{r_1, \dots, r_{|\text{Range}_n^{z_{\text{PILP}}}|}\}$ a partial candidate solution is added to **L** by setting $\bar{x}_n = r_i$. From lemma 4.1, it follows that $x_n^* \in \text{Range}_n^{z_{\text{PILP}}}$, i.e., there exists $\bar{\mathbf{x}}^* \in \mathbf{L}$ such that $\bar{x}_n^* = x_n^*$. The algorithm then analyses all the partial candidate solutions added to **L** and, therefore, after a finite number of steps it considers the candidate $\bar{\mathbf{x}}^*$, and defines $\mathbf{AV} = \{x_1, \dots, x_{n-1}\}$ and $\mathbf{Prob} = \mathbf{PILP}|_{\bar{\mathbf{x}}^*}$. Without loss of generality, let us suppose that $\bar{\mathbf{x}}^*$ is the only partial candidate contained in **L** when it is chosen. The process is then restarted from step 1 until the algorithm terminates. The rest of the proof continues in the same manner as in case 1(b). \square

5 Computational experiments

The performance of the PSA-*pilp* algorithm was compared with that of CPLEX v.12.1.0 (default) on two types of instances randomly generated of the 0-1MKP. Our algorithm was written in C, and the tests were carried out on one core of an Intel i7 3.40GHz with 16 GB of RAM.

5.1 Data generation of test instances

We consider the 0-1MKP, which is stated as follows:

$$(\mathbf{0} - \mathbf{1MKP}) \quad \max \quad z(\mathbf{x}) = \sum_{j=1}^n c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i \in \{1, 2, \dots, m\},$$

with $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$, and c_j, a_{ij} and $b_i \in \mathbb{Z}_+ \quad \forall j \in \{1, \dots, n\}, i \in \{1, \dots, m\}$.

The test instances used in this section were randomly generated following the procedure proposed in Fréville [5]. In all of these instances the coefficients a_{ij} are integer numbers uniformly generated in $U(0, 1000)$; the right-hand side coefficients (b_i 's) are set using the formula $b_i = \alpha \sum_{j \in N} a_{ij}$, where α is the tightness ratio; and the objective function coefficients (c_j 's) are correlated to a_{ij} as follows:

- uncorrelated: $c_j \in U(0, 1000)$
- weakly correlated: $c_j = \frac{\sum_{i=1}^m a_{ij}}{m} + \xi$, with $\xi \in U(-100, 100)$

The test instances were generated by varying combinations of constraints ($m = 3$ up to 5) and variables (from $n = 200$ to $n = 10,000$). The tightness ratio, α , was always fixed to 0.5. For each $n - m$ combination, 5 problems were generated.

5.2 Implementation details of the PSA-*pilp* algorithm

The implementation of the PSA-*pilp* algorithm that was used to carry out the computational experiments reported in this part of the paper presents the following characteristics.

- The two-phase procedure described in Section 3.3 was applied to determine the coefficients $P_j^{\mu p}(0)$ and $P_j^{\mu p}(1)$ for every instance of the 0-1MKP. In addition, the LP problems encountered during this routine were solved using the CPLEX callable library.
- Due to the particular characteristics of the 0-1MKP, the lower projections of every test instance were calculated exactly by means of the following formula: $P_j^{\text{low}}(x_j) = c_j x_j + h$ for all $x_j \in [0, 1]$.
- Every time the condition $|Range_j^{\tilde{z}}| > 1$ for all j such that $x_j \in \mathbf{AV}$ was reached, the active variable corresponding to the largest objective value was selected to split the partial candidate solution being scanned.
- The *last in, first out* strategy was employed to manage the list \mathbf{L} during the execution of the `Inspect_Level` algorithm.

5.3 Results and discussion

Tables 1 and 2 below summarize the results obtained by both solvers on the two types of instances described previously. Columns **CPLEX** and **PSA-*pilp*** report the number of instances solved to optimality by each algorithm, followed by the average runtime (in CPU seconds) of those instances. If the number of instances solved to optimality is less than 5, this indicates that the algorithm failed because it ran out of memory when solving some of the instances. Column **levels** shows the average number of levels scanned by the PSA-*pilp* algorithm until an optimal solution was reached. Column **AV** indicates the percentage of variables that remain *active* after the first iteration of the PSA-*pilp* algorithm at the optimal level. Column **ratio** shows the average CPU time ratio between PSA-*pilp* and CPLEX for solving the given set of instances. Finally, column **memory** indicates the average maximum virtual memory consumption (in megabytes) used by each algorithm (CPLEX/PSA-*pilp*) for solving the given set of instances. In all tests reported in this paper we did not limit the running time nor the memory consumption.

Table 1 computational experiments on *Uncorrelated* instances

n	m	CPLEX	PSA-pilp	levels	AV	ratio	memory (CPLEX/PSA-pilp)
1,000	3	(5) 3 s.	(5) 11 s.	37.6	7.4	3.751	negligible / negligible
5,000	3	(5) 109 s.	(5) 180 s.	13.8	2.6	1.648	negligible / negligible
10,000	3	(5) 830 s.	(5) 1,992 s.	7.2	1.4	2.39	953 / 283
1,000	4	(5) 27 s.	(5) 212 s.	46.8	9.5	7.754	negligible / negligible
3,000	4	(5) 1,473 s.	(5) 4,959 s.	27.4	5.2	3.36	1,001 / 411
5,000	4	(5) 5,358 s.	(5) 9,702 s.	18.6	3.8	1.81	3,233 / 644
10,000	4	(5) 39,957 s.	(5) 32,619 s.	11.2	2.3	0.81	22,193 / 1,765
1,000	5	(5) 108 s.	(5) 1,063 s.	67.6	14.2	9.82	negligible / negligible
3,000	5	(5) 10,044 s.	(5) 38,706 s.	35.2	7.1	3.85	4,595 / 1,826

Table 2 computational experiments on *Weakly Correlated* instances

n	m	CPLEX	PSA-pilp	levels	AV	ratio	memory (CPLEX/PSA-pilp)
2,000	3	(5) 792 s.	(5) 593 s.	8.6	8.1	0.749	620 / 213
3,000	3	(5) 1,311 s.	(5) 697 s.	6.8	5.9	0.53	1,379 / 123
5,000	3	(5) 3,704 s.	(5) 1,413 s.	4	3.4	0.381	4,369 / 185
10,000	3	(5) 5,226 s.	(5) 2,971 s.	2.8	2.2	0.568	6,709 / 318
200	4	(5) 35 s.	(5) 228 s.	56.6	56.8	6.551	negligible / negligible
500	4	(5) 355 s.	(5) 988 s.	29.4	30	2.783	negligible / negligible
1,000	4	(5) 5,567 s.	(5) 8,332 s.	20.8	20.4	1.49	3,953 / 905
2,000	4	(5) 32,214 s.	(5) 19,503 s.	11.2	11.1	0.60	24,509 / 2,035
3,000	4	(2) >59,931 s.	(5) 58,529 s.	8.6	8.3	<0.976	>63,658 / 3,502
200	5	(5) 202 s.	(5) 1,821 s.	69	70	9.021	negligible / negligible
500	5	(5) 9,234 s.	(5) 29,591 s.	37.2	38	3.20	4,955 / 2,036
1,000	5	(5) 89,542 s.	(5) 93,581 s.	25.8	26.2	1.04	41,244 / 4,935

Based on the computational results, we conclude that the PSA-pilp algorithm is not very efficient, in terms of running time, to solve small-size instances; however, it shows a better trend than CPLEX (see ratio) when the number of variables increases, especially in the hardest type of instances. In this regard, it is worth noting that, in contrast to CPLEX (default), the implementation of the PSA-pilp algorithm does not incorporate any type of presolve, cutting plane technique, or heuristics to improve its performance. Concerning memory usage, the numbers of the PSA-pilp algorithm are considerably lower than those of CPLEX in all instances tested. The PSA-pilp algorithm consumed in average less than 10.4% of the memory consumed by CPLEX. This can be explained by the way the algorithm conducts the search process for the optimal solution (by generating candidate solutions tailored to specific values of the objective function), and by the manner in which the \mathbf{L} set is managed during the execution of the `InspectLevel` algorithm. In fact, under these conditions it can be proven that PSA-pilp's memory consumption is polynomial in the number of variables and the cardinality of the sets $Range_j^z$. Finally, it is interesting to note that the percentage of variables that are fixed to their optimal value in the first iteration of the algorithm at the optimal level, grows to more than 97% of the total variables.

6 Conclusions and future work

This paper proposes a new exact algorithm, called PSA-pilp, for solving PILP problems using projections. The PSA-pilp algorithm differs from state-of-the-art techniques since it searches for solutions for specific values of the objective function. As a consequence of this approach, the number of variables in the original problem is

systematically reduced (for each considered level) and no additional constraints are added to the initial formulation. According to our computational experiments, we believe that the proposed new algorithm paradigm has a great potential as a useful tool for solving PILP problems.

The present work leaves open a number of interesting directions for future research. First, the current version of the PSA-*pilp* algorithm could be greatly improved through the incorporation of advanced search strategies, preprocessing and probing techniques, cutting plane algorithms, and primal heuristics. Second, additional improvements can be reached via parallel computing techniques. Projection-splitting-based algorithms are natural candidates for parallelization because the subproblems associated with each level and each partial candidate solution contained in \mathbf{L} are completely independent. Thus, parallelism can be exploited by evaluating multiple levels and multiple partial candidate solutions simultaneously. Finally, it is relatively easy to see how the proposed methodology can be extended to more complex situations such as PILP problems in which the condition $\mathbf{c} \in \mathbb{Z}^n$ and $h \in \mathbb{Z}$ in the objective function is relaxed, or even to MILP problems. We are going to deal with this discussion in the second part of this series.

Acknowledgements This work was partially supported by grants UBACYT 20020100100666, PICT 2010-304, PICT 2011-817. We thank Luis Mastrangelo and Santiago Feldman for their helpful suggestions and constructive criticisms.

References

- [1] Chvátal, V.: Linear Programming, W.H. Freeman (1983)
- [2] IBM ILOG CPLEX Optimizer, <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/> (2014). Accessed 1 April 2014
- [3] Dantzig, G.B.: Linear Programming and Extensions. Elsevier (2004)
- [4] Dantzig, G.B., Thapa, M. N.: Linear Programming, 1: Introduction. Princeton University Press (1963)
- [5] Fréville, A.: The multidimensional 0-1 knapsack problem: an overview. Elsevier (2004)
- [6] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco (1979)
- [7] Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society 64, 275-278 (1958)
- [8] Gurobi Optimizer, <http://www.gurobi.com/> (2014). Accessed 1 April 2014
- [9] Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica 4, 373-395 (1984)
- [10] Land, A.H., Doig, A.G.: An Automatic Method for Solving Discrete Programming Problems. Econometrica 28, 497-520 (1960)
- [11] Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons (1990).
- [12] Nemhauser, G., Wolsey, L.: Integer Programming and Combinatorial Optimization. Wiley (1988)
- [13] Wolsey, L.: Integer Programming. Wiley (1998)